In use, the assembly-language routine must save the return address, and must push it on the stack again just before returning to the calling program. The passed parameters are available on the stack in the reverse order to the order in which they were originally pushed on the stack.

The conventions of the surrounding system concerning register use and calling sequences must be respected by writers of assembly-language routines. On the Apple, all registers are available, and zero-page hexadecimal locations Ø through 35 are available as temporary variables. However, the Apple Pascal system also uses these locations as temporaries, so you should not expect data to remain there from one execution of a routine to the next. You can save variables in non-zero page memory by using the .BYTE or .WORD directives in your routine to reserve space.

For external assembly-language functions (.FUNC's) only, two additional conventions must be recognized:

1) At the function's entry time, the Pascal host program pushes two words (four bytes) of zeros on the evaluation stack after any passed parameters are put on the stack and before the return address is pushed on the stack.

2) At the function's exit time, the .FUNC must push the function result (a scalar, real, or pointer, maximum two words), high byte first, just before pushing the return address on the stack.

For an example of an external assembly-language procedure, an external assembly-language function, and a Pascal host program which calls these routines, see the EXAMPLE earlier in this chapter. The EXAMPLE also demonstrates the handling of the return address, passed parameters, and returned function value in assembly-language routines. The external routines in that example are manually Linked into the Pascal calling program. For information about installing a routine into the system library, see this manual's chapter UTILITY PROGRAMS.

THE ASSEMBLER DIRECTIVES

AN OVERVIEW

Assembler directives (also referred to as "pseudo-ops") let you tell the Assembler to do various functions other than provide directly executable code. The following directives are common to all versions of the UCSD Adaptable Assembler, including the Apple Pascal 6502 Assembler, but may differ from individual manufacturer's standard syntax.

In the following descriptions of directives, square brackets [like this] are metasymbols that denote optional elements which you may supply. Angle brackets <like this> are meta-symbols that denote required elements which you must supply. If an element type is not shown, it cannot be used in that situation.

EXAMPLE:

```
        [label]    .ASCII    "<character string>"
```

This notation indicates that you may supply a label, but it is not necessary, and that between the required double quotes you must supply the character string to be converted (not necessarily the words "character string"). The bracket metasymbols are not to be be typed.

The following terms represent general concepts in the explanation of each directive:

TERM:	DEFINITION:
value	Any numerical value, label, constant, or expression.
valuelist	A list of one or more values separated by commas.
identifierlist	A list of one or more identifiers separated by commas.
expression	Any legal expression as defined under SYNTAX OF ASSEMBLY STATEMENTS.
identifier[:integer] list	A list of one or more identifier:integer pairs separated by commas. The colon-integer is optional in each pair and the default is 1.

Small examples are included after each directive definition to show you the specific syntax and form of that directive. The EXAMPLE assembly-language routine earlier in this chapter is used to show the combined use and detailed examples of directive operations.

ROUTINE-DELIMITING DIRECTIVES

Every assembly must include at least one .PROC or .FUNC, and one .END , even in the case of stand-alone code which will not be linked into a Pascal host (e.g., the interpreter). The most frequent use of the Assembler, however, will be small routines intended to be linked with a Pascal host. In this case, .PROCs and .FUNCs are used to identify and delimit the assembly code to be accessed by a Pascal external procedure or function. The .END appears at the end of the last routine and serves as the final delimiter.

References to an assembly-language .PROC or .FUNC are made in the Pascal host program by use of EXTERNAL declarations. At the time of this declaration the actual parameter names must be given. For example, if the Pascal host's declaration is:

```
          PROCEDURE FARKLE(X,Y:REAL);
          EXTERNAL;
```

the associated declaration for the assembly-language .PROC would be

```
                    .PROC FARKLE,4
```

A .PROC, .FUNC, or any assembly routine should be inserted into the SYSTEM.LIBRARY so that it can be referenced by the Linker and linked into the Pascal host program at R(un time. An alternate method would be to execute the Linker and tell it what files to link in. Either method works. However, if the Pascal host is updated and the assembly routines have not been installed in the SYSTEM.LIBRARY, the Linker will have to be executed again after each host program update. Therefore, we suggest that the routines be inserted into the SYSTEM.LIBRARY to avoid this repetition. If the Linker is called automatically, using the R(un command, it will automatically search the SYSTEM.LIBRARY for the appropriate definition of the assembly routine and link the two together.

The EXAMPLE earlier in this chapter shows the use of assembly-language routines from a Pascal host program and demonstrates the manual linking process. More information on linking appears in this manual's chapter THE LINKER. For information on using the system librarian to install a routine into SYSTEM.LIBRARY , see this manual's chapter UTILITY PROGRAMS.

.PROC   Identifies a procedure that returns no value. A .PROC is ended by the occurrence of a new .PROC , .FUNC , or .END .

```
         FORM:          .PROC  <identifier>[,expression]

                        [expression] indicates the number of words
                        of parameters expected by this routine.
                        The default is 0.

         EXAMPLE:       .PROC   DLDRIVE,2
```

.FUNC   Identifies a function that returns a value. Two words of space to be used for the function value will be placed on the stack after any parameters. A .FUNC is ended by the occurrence of a new .PROC , .FUNC , or .END .

```
         FORM:          .FUNC  <identifier>[,expression]

                        [expression] indicates the number of words
                        of parameters expected by this routine.
                        The default is 0.

         EXAMPLE:       .FUNC  RANDOM,4
```

.END   Used to denote the physical end of an assembly.

```
         FORM:          .END

         EXAMPLE:       .END
```

LABEL DEFINITIONS AND SPACE ALLOCATION DIRECTIVES

.ASCII   Converts character values to ASCII equivalent byte constants and places the equivalents into the code stream.

```
         FORM:    [label]  .ASCII    "<character string>"
```

where <character string> is any string of printable ASCII characters, including a space. The length of the string must be less than 80 characters. The double quotes are used as delimiters for the characters to be converted. If a double quote is desired in the string, it must be specifically inserted using a .BYTE .

```
         EXAMPLE:       .ASCII    "HELLO"
```

for the insertion of AB"CD the code must be constructed as:

```
                        .ASCII    "AB"
                        .BYTE      22       ; An ASCII  "
                        .ASCII    "CD"
```

Note: The 22 is the hexadecimal ASCII code for a double quote.

.BYTE Allocates a byte of space into the code stream for each value listed. Each value actually stored by the routine must have a value between -128 and +255. If the value is outside of this range an error will be flagged. Assigns the associated label, if any, to the address at which the byte was stored.

```
FORM:     [label]  .BYTE   [valuelist]

          the default for no stated value is Ø.

EXAMPLE:  TEMP  .BYTE    4

          the associated output would be:  Ø4
```

.BLOCK Allocates a block of space into the code stream for each value listed. Amount allocated is in bytes. Associates the label (if present) with the starting address of the block allocated.

```
FORM:    [label]  .BLOCK   <length>[,value]
```

<length> is the the number of bytes to hold the <value> specified. The default for no stated value is Ø.

```
EXAMPLE:  TEMP  .BLOCK    4,6

          the associated output would be:

             Ø6
             Ø6          ( four bytes with the value Ø6 )
             Ø6
             Ø6
```

.WORD Allocates a word of space in the code stream for each value in the valuelist. Associates the declaration label with the word space allocation.

```
FORM:    [label]  .WORD   <valuelist>

EXAMPLE:  TEMP  .WORD   Ø,2,4,...

          the associated output would be:

             ØØØØ
             ØØØ2
             ØØØ4   (words with these values in them)
               .
               .
```

```
EXAMPLE:   A1    .WORD  A2
                   .
                   .
                   .
           A2    .EQU   $          ;   $ denotes LC value
                 .WORD  5.
                   .
```

The statement A2 .EQU $ assigns the current value of the location counter (LC) to the label A2 . If the value of the location counter is 5Ø at the .EQU , the associated output would be:

```
     ØØ5Ø  ( assignment due to the value of L2 )
      .
      .
     ØØØ5  ( assignment due to the .WORD 5 )
      .
```

.EQU Assigns a value to a label. Labels may be equated to an expression containing labels and/or absolutes. One must define a label before it is used unless it will simply be equated to another label. A local label may not appear on the left-hand side of an equate (.EQU).

```
FORM:     <label>  .EQU     <value>

EXAMPLE:    BASE   .EQU     R6
```

.ORG Takes the operand of .ORG as the offset, relative to the start of the assembly file, where the next word or byte of code is to go. Words or bytes of zeros are produced to get the current location counter (LC) to the correct value.

```
FORM:               .ORG      <value>

EXAMPLE:            .ORG      ØDØØØ
```

.ABSOLUTE If a .ABSOLUTE occurs before the first .PROC then all .ORG's are interpreted as absolute memory locations. The user must take responsibility for the correct loading of the produced code file. The use of .ABSOLUTE has the effect of cancelling the generation of relocation information. Further, any defined (i.e., non forward-referenced) labels may be treated as absolute numbers. Thus such labels may be multiplied and divided, etc. .ABSOLUTE must occur before the first .PROC and is set for the entire assembly.

```
FORM:               .ABSOLUTE

EXAMPLE:            .ABSOLUTE
```

.INTERP Interpreter relative locations are specified by the use of .INTERP in an expression. Further labels may defined as interpreter relative in the manner shown in the example. The rules regarding the use of such labels are the same as for any other specially defined labels (e.g., .PUBLIC and .PRIVATE). Locations whose values depend on interpreter relative labels or expressions are listed in a fourth relocation list at the end of the assembly procedure.

```
EXAMPLE:    STUFF .EQU   .INTERP+25
```

Certain interpreter entry points may be useful, using an instruction such as

```
        LDA    @.INTERP+n
```

with these values of n:

n=Ø   Address of the execution error routine; displays error message using the error number in the A register.

n=2   Address of the BIDS jump table; handles input and output.

n=4   Address of SYSCOM; system's communications area of the P-machine.

MACRO FACILITY DIRECTIVES

A macro is a named section of text that can be defined once and repeated in other places simply by using its name. The text of the macro may be parameterized, so that each invocation results in a different version of the macro contents. The entire macro definition may precede the first .PROC or .FUNC of the assembly file.

At the invocation point, the macro name is followed by a list of parameters, each terminated by a comma (except for the last one, which is terminated by end of line or the comment indication (;). The text of the macro definition, modified by substituting the invocation parameters, is inserted (conceptually speaking) by the Assembler at the invocation point. Wherever %n (where n is a single decimal digit greater that zero) occurs in the macro definition, the text of the n-th invocation parameter is substituted. Leading and trailing blanks are stripped from the parameter before the substitution. If the macro definition includes a reference to a parameter not provided in a particular invocation (too few parameters or no parameter before a terminating comma), a null string is substituted.

A macro definition may not contain another macro definition. A definition can certainly, however, include macro invocations. This "nesting" of macro invocations is limited to five levels deep.

The expanded macro is always included in the listing file (unless .NOMACROLIST is in effect at the point of invocation). Macro expansion text is flagged, in the listing, by a # just left of each

expanded line. Comments occurring in the macro definition are not repeated in the expansion.

.MACRO   Indicates the start of a macro definition and gives it an identifier.

.ENDM   Indicates the end point of a macro definition.

```
FORM:              .MACRO      <identifier>
                     .
                     .                        ; (macro body)
                     .
                   .ENDM

EXAMPLE:           .MACRO HELP
                   STA    %1                  ; < comment >
                   LDA    %2                  ; < comment >
                   .ENDM
```

The assembly listing beginning at the point where this macro was invoked may look like this:

```
             HELP  ALPHA,BETA
        #    STA   ALPHA
        #    LDA   BETA
```

The statement HELP calls the defined macro and sends it two parameters, ALPHA and BETA. These parameters are in turn used in forming the macro expansion (flagged in the listing by # signs) that follows the invoking statement. In the expansion, the first calling-statement parameter (variable ALPHA) is substituted for the definition's identifier %1 , and the second parameter (variable BETA) is substituted for the identifier %2 .

The following portion of an assembled listing illustrates the syntax used when defining and invoking macros. The procedure itself is not meant to be an actual, useful program.

```
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
 PAGE -   1  TEMP2     FILE:MACROCALL

ØØØØ|                          .PROC TEMP2 ; SHOWS SYNTAX OF MACRO CALLS
Current memory available:   1ØØ88
ØØØØ|                                     ; CONSTANTS
ØØØØ|
ØØØØ| ØØØA              CON1Ø    .EQU  1Ø.
ØØØØ| ØØBF              OTHØ     .EQU  ØBFH
ØØØØ| ØØF7              ONEØ     .EQU  ØF7H
ØØØØ|
ØØØØ|                                     ; MACRO DEFINITIONS
ØØØØ|
ØØØØ|                            .MACRO M2
ØØØØ|                              CLC
ØØØØ|                              LDA    PREDEFL+%1
```

```
ØØØØ|                                    .ENDM
ØØØØ|
ØØØØ|                                    .MACRO TESTM
ØØØØ|                                      JMP  %1
ØØØØ|                                      LDA  #5+%2
ØØØØ|                                      M2   %2 ; MACRO CALL WITHIN A MACRO DEF'N
ØØØØ|                                      LDA  %3
ØØØØ|                                      LDA  %4
ØØØØ|                                      LDA  %5
ØØØØ|                                      JMP  %6
ØØØØ|                                    .ENDM
ØØØØ|
ØØØØ| A5 Ø5                    PREDEFL   LDA  5  ; A PRE-DEFINED LABEL
ØØØ2|
ØØØ2|                                              ; MACRO CALL WITH ALL PARAMETERS
ØØØ2|                                              ; & NO LEADING OR TRAILING SPACES
ØØØ2|
ØØØ2|                                     TESTM PREDEFL,<5*CON1Ø+6>,#55,#6,1,LABEL2
ØØØ2| 4C ØØØØ                  #          JMP  PREDEFL
ØØØ5| A9 3D                    #          LDA  #5+<5*CON1Ø+6>
ØØØ7|                          #          M2   <5*CON1Ø+6>
ØØØ7| 18                       #          CLC
ØØØ8| AD 38ØØ                  #          LDA   PREDEFL+<5*CON1Ø+6>
ØØØB| A9 55                    #          LDA  #55
ØØØD| A9 Ø6                    #          LDA  #6
ØØØF| A5 Ø1                    #          LDA  1
ØØ11| 4C ****                  #          JMP  LABEL2
ØØ14|
ØØ14|                                     M2   5  ; SIMPLE MACRO CALL
ØØ14| 18                       #          CLC
ØØ15| AD Ø5ØØ                  #          LDA   PREDEFL+5
ØØ18|
ØØ18|                                              ; MACRO CALL WITH NUL PARAMETERS
ØØ18|                                              ; AND LEADING & TRAILING SPACES
ØØ18|
ØØ18|                                     TESTM  ,CON1Ø,,   XX  ,ØFØH, PREDEFL

       JMP
not enough operands
E(dit,<space>,<esc>      [ Spacebar pressed here, to continue assembly. ]

ØØ18|                          #          JMP
ØØ18| A9 ØF                    #          LDA  #5+CON1Ø
ØØ1A|                          #          M2  CON1Ø
ØØ1A| 18                       #          CLC
ØØ1B| AD ØAØØ                  #          LDA   PREDEFL+CON1Ø
```

```
       LDA
ill formed operand
E(dit,<space>,<esc>      [ Spacebar pressed here, to continue assembly. ]

ØØ1E|                    #      LDA
ØØ1E| AD ****            #      LDA  XX
ØØ21| A5 FØ              #      LDA  ØFØH
ØØ23| 4C ØØØØ            #      JMP  PREDEFL
ØØ26|
ØØ26|                         .END
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
```

CONDITIONAL ASSEMBLY DIRECTIVES

Conditionals are used to selectively exclude or include sections of code at assembly time. When the Assembler encounters a .IF directive, it evaluates the associated expression. In the simplest case, if the expression is false, the Assembler simply discards the text until a .ENDC is reached. If there is a .ELSE directive between the .IF and .ENDC directives, the text before the .ELSE is selected if the expression is true, and the text after the .ELSE if the condition is false. The unassembled part of the conditional will not be included in any listing. Conditionals may be nested.

The conditional expression takes one of two forms. The first is the normal arithmetic/logical expression used elsewhere in the Assembler. This type of expression is considered false if it evaluates to zero; true otherwise. The second form of conditional expression is comparison for equality (indicated by =) or inequality (indicated by <>). One may compare strings, characters, or arithmetic/logical expressions.

.IF   Identifies the beginning of the conditional.

.ENDC   Identifies the end of a conditional .IF

.ELSE   Identifies the alternate to the .IF If the conditional expression is equal to Ø then the else portion is used.

```
FORM:   [label]  .IF  <expression>
                  .
                  .
                  .
              [ .ELSE ]
                  .
                  .          ; (only if there is an else)
                  .
                .ENDC
```

where the expression is the conditional expression to be met.

EXAMPLE:

```
        .IF LABEL1-LABEL2      ;Arithmetic expression.
         .                     ; This text assembled
         .                     ; only if subtraction
         .                     ; result is non-zero
         .

        .IF  "%1" ="STUFF"     ;Comparison expression.
         .                     ; This text assembled
         .                     ; if subtraction above
         .                     ; was true and if text
         .                     ; of first parameter
         .                     ; (assume we're in macro)
         .                     ; is equal to "STUFF".
        .ENDC                  ; Terminates nested cond.
         .
         .
        .ELSE
         .                     ; This text assembled if
         .                     ; subtraction result was
         .                     ; zero.
         .
        .ENDC                  ; Terminates outer level
                               ; of conditional.
```

HOST-COMMUNICATION DIRECTIVES

The directives .CONST , .PUBLIC , and .PRIVATE allow the sharing of information and data space between an assembly routine and the host program which uses that routine. These external references must eventually be resolved by the Linker. Refer to this manual's chapter THE LINKER for further details.

.CONST Allows globally declared constants in the host program to be accessed by the assembly routine. .CONST can only be used in a program to replace 16-bit relocatable objects.

FORM: .CONST <identifierlist>

EXAMPLE: (see example after .PRIVATE)

.PUBLIC Allows a variable declared in the global data segment of the host program to be used by both the assembly-language routine and the host program.

FORM: .PUBLIC <identifierlist>

EXAMPLE: (see example after .PRIVATE)

.PRIVATE Allows variables of the assembly routine to be stored in the host program's global data segment and yet be inaccessible to the host program. These variables retain their values for the entire execution of the program.

FORM: .PRIVATE <identifier[:integer] list>

The integer is used to communicate the number of words to be allocated to the identifier. The default is one word.

EXAMPLE: (for .CONST, .PRIVATE, and .PUBLIC)

Given the following Pascal host program:

```
PROGRAM EXAMPLE;
CONST SETSIZE=5Ø; LENGTH=8Ø;

VAR I,J,F,HOLD,COUNTER,LDC:INTEGER;
    LST1:ARRAY[Ø..9] OF CHAR;

BEGIN
  .
  .
  .
END.
```

and the following section of an assembly routine:

```
.CONST      LENGTH
.PRIVATE    PRT,LST2:9
.PUBLIC     LDC,I,J
```

This will allow the constant LENGTH to be used in the assembly routine almost as if the line LENGTH .EQU 8Ø. had been written. (Recall the limitation mentioned above for using .CONST identifiers.) The variables LDC,I and J are to be used by both the Pascal host and the assembly routine, while the variables PRT and LST2 are to be used only by the assembly routine. Further, the LST2:9 causes the variable LST2 to correspond with the beginning of a nine-word block of space in the Pascal host's global data segment.

EXTERNAL REFERENCE DIRECTIVES

Separate routines may share data structures and subroutines by linkage from one assembly routine to another assembly routine. This is made possible through the use of .DEF and .REF . These directives cause the Assembler to generate link information that allows two separately assembled routines to be linked together. By using .DEF and .REF , one assembly routine may call subroutines found in another assembly routine. One routine placed in a library file such as the boot diskette's SYSTEM.LIBRARY can contain a large number of frequently used subroutines which are all available to other routines.

The use of .DEF and .REF is similar to that of .PUBLIC . .DEFs and .REFs associate labels between two assembly routines rather than between an assembly routine and a Pascal host program. Just as with .PRIVATE and .PUBLIC , these external references must eventually be resolved by the Linker. If such resolution cannot be accomplished, the Linker will indicate the offending label. Naturally, the Assembler cannot be expected to flag these errors, since it has no knowledge of other assemblies.

The host assembly routine must be linked to its external assembly subroutines BEFORE that host assembly routine can be linked into a Pascal host program or UNIT as an EXTERNAL procedure or function.

.DEF

Identifies a label that is defined in the current routine as being available for use (by means of .REF) from .PROCs or .FUNCs in other assembly-language routines.

Note: The .PROC and the .FUNC directives also generate a .DEF with the same name. This allows a host assembly routine to call external .PROCs and .FUNCs if the host assembly routine has defined them in a .REF .

FORM: .DEF <identifierlist>

EXAMPLE: The following sketched-out routine declares a .DEF for the labels DOIT and THINK . The subroutines bearing the labels DOIT and THINK may then be used by other assembly routines (see example for .REF).

```
        .PROC FARKLE,3
        .DEF DOIT,THINK
           .
           .
           BNE THINK
           .
DOIT       LDA
           .
           RTS
           .
THINK      LDY
           .
           RTS
           .
           .END
```

.REF

Identifies a label used in the current routine which refers to a label declared as available (by means of .DEF) in another routine's .PROC or .FUNC . During the linking process, corresponding .DEFs and .REFs are matched.

Note: The .PROC and the .FUNC directives also generate a .DEF with the same name. This allows a host assembly routine to call external .PROCs and .FUNCs if the host assembly routine has defined them in a .REF .

FORM: .REF <identifierlist>

EXAMPLE: The following sketched-out assembly-language routine declares a .REF for the external label DOIT (DOIT was declared available for such reference by the .DEF in the previous example). It then uses that label just as if it referred to a labelled subroutine within the routine itself.

```
.PROC SAMPLE
.REF DOIT
  .
  .
JSR DOIT
  .
  .
.END
```

Note: The assembly routine containing .PROC FARKLE must be linked from its library codefile into the host assembly routine containing .PROC SAMPLE before SAMPLE can be linked in as an EXTERNAL procedure to a Pascal UNIT or program.

LISTING CONTROL DIRECTIVES

The listing control directives determine what is sent to the output file that is specified at assembly time, in response to the prompt

OUTPUT FILE FOR ASSEMBLED LISTING: (<CR> FOR NONE)

If no listing output file is specified (by just pressing the RETURN key), then all listing control directives are simply ignored as irrelevent.

.LIST and .NOLIST

Allows selective listing of assembly routines. Listing goes to the specified output file when a .LIST is encountered. The .NOLIST is used to turn off the .LIST option. Listing may be turned on and off repeatedly within an assembly. .LIST is the default state.

```
FORM:          .LIST     or     .NOLIST
```

.MACROLIST and .NOMACROLIST

Allows selective listing of macro expansions. In general an assembled listing will contain the textual expansion of a macro if the .MACROLIST option was in effect when the macro was defined. On the other hand, an assembled listing will not contain the textual expansion of a macro if the .NOMACROLIST option was in effect when the macro was defined. These options may be used repeatedly throughout an assembly, to list the expansions of certain macros selectively.

Macro expansion text is flagged in the listing by a # to the left of each expanded line. Comments occurring in the macro definition are not repeated in the expansion. The assembled listing of the EXAMPLE earlier in this chapter shows the macro POP defined on PAGE-Ø , and listings of the macro expansion appear on PAGE-1 and PAGE-4 .

When assembling nested macro invocations, listing of textual expansion continues until the Assembler encounters the first macro defined with .NOMACROLIST in effect. Listing does not resume until that macro's invocation is complete, regardless of the listing state of the macros invoked by the non-listing macro.

The .LIST and .NOLIST options take precedence over the .MACROLIST and .NOMACROLIST options. The Assembler defaults to the .MACROLIST state.

```
FORM:          .MACROLIST     or     .NOMACROLIST

EXAMPLE:       .NOMACROLIST
```

.PATCHLIST and .NOPATCHLIST

Allow control over listing of back-patches made to the code file. These options may be used repeatedly throughout an assembly.

When an undefined label is encountered, the assembled listing shows one * for each hexadecimal digit to be filled in later. For example:

```
ØØ19| 1Ø**            BPL DONE
```

When the forward reference is resolved, the back-patch is listed in the form

```
ØØ19* ØØ
ØØ1F| A9 ØØ      DONE  LDA #Ø
```

where the number to the left of the asterisk is the address of the patched location and the number to the right of the asterisk is that location's new value. See PAGE-1 of the assembled listing of the EXAMPLE, earlier in this chapter, for an illustration of back-patch listing.

.PATCHLIST is the default state.

```
FORM:          .PATCHLIST     or     .NOPATCHLIST

EXAMPLE:       .NOPATCHLIST
```

.PAGE

Allows the programmer to explicitly ask for a top of form page break in the listing.

```
FORM:          .PAGE

EXAMPLE:       .PAGE
```

.TITLE

Allows the titling of each page if desired. At the start of each procedure the title is set to blanks and must be reset i title is desired. The title is only cleared at the start of the file. In the EXAMPLE assembly listing earlier in this chapter, the title SYMBOLTABLE DUMP was not set by a .TITLE directive. That heading is always used on pages containing symboltable dumps. Upon assembling a further procedure the heading printed returns to what it was before the symboltable dump.

```
FORM:          .TITLE "<title>"
```

where <title> is any string of printable ASCII characters, including a space. The length of the string must be less than 8Ø characters. The double quotes are used as delimiters for the string, so a title may not include the double quote character.

```
EXAMPLE:       .TITLE  "QRC12 INTERPRETER"
```

FILE DIRECTIVE

```
.INCLUDE   Causes the indicated source file to be included at that point.

           FORM:              .INCLUDE   <filename>

                      where the filename specifies an assembly-
                      language textfile to be included.
```

If you don't add the suffix .TEXT the system will add it for you. The last character of the filename must be the last non-space character on that line (no comment may follow on the same line).

```
CORRECT EXAMPLE:     .INCLUDE   SHORTSTART.TEXT

CORRECT EXAMPLE:     .INCLUDE   SHORTSTART.TEXT
                                ; CALLS STARTER

INCORRECT EXAMPLE:   .INCLUDE   SHORTSTART.TEXT   ; CALLS STARTER
```

The Include-file's text is treated by the assembler just as if you had typed that text into the original file at the position of the .INCLUDE directive. For example, if the included file contains a .END , the assembly is terminated at that point.

Note: For a list of Assembler error messages, see the appendix at the end of this manual.

ASSEMBLER DIRECTIVE SUMMARY

METASYMBOL NOTATION

Square brackets [like this] surround optional elements which you may supply. Angle brackets <like this> surround required elements which you must supply. The metasymbol brackets and the brief definition at the end of each line are not to be typed.

ROUTINE DELIMITING DIRECTIVES

```
.PROC   <identifier>[,expression]   Begins a procedure.
.FUNC   <identifier>[,expression]   Begins a function.
.END                                Ends entire assembly.
```

LABEL DEFINITIONS AND SPACE-ALLOCATION DIRECTIVES

```
[label]   .ASCII   "<character string>"   Inserts ASCII of chars.
[label]   .BYTE    [valuelist]            Inserts byte of value.
[label]   .BLOCK   <length>[,value]       Inserts block of value.
[label]   .WORD    <valuelist>            Inserts word of value.
<label>   .EQU     <value>                Assigns value to label.
          .ORG     <value>                Next byte at start of
                                            assembly file + value.
          .ABSOLUTE                       Precedes 1st .PROC; all
                                            .ORGs put next byte at
                                            abs. location = value.
          .INTERP                         1st loc. of interpreter,
                                            in relative-location
                                            expressions.
```

MACRO FACILITY DIRECTIVES

```
          .MACRO   <identifier>           Begins a macro definition.
          .ENDM                           Ends a macro definition.
```

CONDITIONAL ASSEMBLY DIRECTIVES

```
[label]   .IF      <expression>           Begins condit'l assembly.
                                          If true, assembles next
        [ .ELSE ]                            text [up to .ELSE ];
                                           if false, only text
                                           after a .ELSE .
          .ENDC                           Ends condit'l assembly.
```

HOST-COMMUNICATION DIRECTIVES

```
        .CONST   <identifierlist>         Takes value from global
                                            const in Pascal host.
        .PUBLIC  <identifierlist>         Uses a global variable
                                            from the Pascal host.
        .PRIVATE <identifier[:integer] list>  Variable not accessible
                                            to the Pascal host.
                                            Default :1 word/ident.
```

EXTERNAL COMMUNICATION DIRECTIVES

```
          .DEF     <identifierlist>       Makes label available
                                            to other routines.
          .REF     <identifierlist>       Label refers to another
                                            routine's .DEF'd label.
```

LISTING CONTROL DIRECTIVES

.LIST and .NOLIST	Turns assembly listing on and off.
.MACROLIST and .NOMACROLIST	Turns listing of macro expansions on and off.
.PATCHLIST and .NOPATCHLIST	Turns listing of back-patches on and off.
.PAGE	Puts page-feed in listing.
.TITLE "<title>"	Titles each page of current .PROC or .FUNC .

FILE DIRECTIVE

.INCLUDE <filename>	Includes named text file in the assembly.

Note: Additional information can be found in this manual's chapters THE LINKER (Linker information), UTILITY PROGRAMS (installing routines in SYSTEM.LIBRARY), and in the TABLES appendix (Assembler error messages).

CHAPTER 7 THE LINKER